NASA Technical Memorandum 103671

Efficient Computation of Aerodynamic Influence Coefficients for Aeroelastic Analysis on a Transputer Network

David C. Janetzke
Lewis Research Center
Cleveland, Ohio

and

Durbha V. Murthy
The University of Toledo
Toledo, Ohio

Prepared for the
Symposium on Parallel Methods on Large-scale Structural Analysis and Physics Applications
cosponsored by National Aeronautics and Space Administration Langley Research Center and United States Air Force Weapons Laboratory
Hampton, Virginia, February 5-6, 1991.

EFFICIENT COMPUTATION OF AERODYNAMIC INFLUENCE COEFFICIENTS FOR AEROELASTIC ANALYSIS ON A TRANSPUTER NETWORK

David C. Janetzke
NASA Lewis Research Center
Cleveland, Ohio 44135

and

Durbha V. Murthy*
The University of Toledo
Toledo, Ohio 43606

Abstract

Aeroelastic analysis is multi-disciplinary and computationally expensive. Hence, it can greatly benefit from parallel processing. As part of an effort to develop an aeroelastic analysis capability on a distributed memory transputer network, a parallel algorithm for the computation of aerodynamic influence coefficients is implemented on a network of 32 transputers. The aerodynamic influence coefficients are calculated using a 3-dimensional unsteady aerodynamic model and a panel discretization. Efficiencies up to 85 percent were demonstrated using 32 processors. The effects of subtask ordering, problem size and network topology are presented. A comparison to results on a shared memory computer indicates that higher speedup is achieved on the distributed memory system.

Nomenclature

A	area on blade surface
B	proportionality constant defined in eq. (1)
c_{ij}	i, j-th element of C
C	aerodynamic influence coefficient matrix
dA_j	area of the j -th panel
D	defined in eq. (6)
f_n	generalized motion-independent aerodynamic force
K	kernel function
m, n	assumed mode shape indices
M	Mach number
P, P_0	points on the blade surface
Q	generalized motion-dependent aerodynamic force
r	radial coordinate
W	normalized normal velocity on blade surface
δ	normal displacement

---

* NASA Resident Research Associate at Lewis Research Center.

$\Delta\bar{p}$	normalized unsteady pressure disturbance
ω	vibration frequency
Ω	rotational speed
$\bar{\theta}$	azimuthal coordinate

Subscripts

0	dummy variable
i, j	panel indices
LE	leading edge
TE	trailing edge

Introduction

The demands of light structural weight, extreme flexibility and high flight dynamic pressure on modern high-performance atmospheric vehicles lead to several undesirable phenomena of aeroelastic nature. In addition to wings and propellers, compressor and turbine blades, fan blades, flow guide vanes and civil engineering structures like bridges and buildings are also subject to aeroelastic phenomena. Thus, aeroelastic analysis is of great importance in aeronautical, mechanical and civil engineering. It is the topic of many books and research papers.

Undesirable aeroelastic phenomena are generally classified into three categories: static aeroelastic phenomena, flutter and forced response. The first, static aeroelastic phenomena, lead to failure by divergence and control surface reversal. The other two encompass dynamic phenomena and lead to failure by flow-induced vibration. Flutter failure occurs when such vibrations are self-excited and thus unstable. Flutter is a result of the motion-dependent unsteady aerodynamic forces which are out of phase with structural motion. On the other hand, aeroelastic forced response could induce flow-induced fatigue failure due to stable but significantly high amplitude vibrations. Such vibrations are a result of the dynamic response of the structure modified by the presence of motion-independent unsteady aerodynamic forces.

Aeroelastic analysis is by definition multi-disciplinary in nature and involves the coupling of dynamic structural analysis and unsteady aerodynamic analysis. For all but the simplest models, both the components of the analysis are computationally expensive and require state-of-the-art computational resources to obtain results in a reasonable amount of time. Current sequential processing computers are inadequate for routine aeroelastic analysis of high-performance propellers and turbomachinery blades using advanced structural and aerodynamic models. The computational burden is particularly large if the aeroelastic analysis is to be performed many times, such as inside a design iteration loop

or an optimization program. The computational power of the new parallel processing computer systems offers the aeroelastician an opportunity to reduce the computational times so that the aeroelastic analysis can be more fruitfully used by the designers. As part of an effort in this direction, a research program at NASA Lewis Research Center involves the concurrent processing adaptation of advanced aeroelastic analysis codes for rotating systems.

There is already a large body of literature regarding the parallelization of structural dynamic analysis using finite element methods (e.g. see references in Noor and Atluri, 1987; Noor and Venneri, 1990). The present study is only concerned with the parallelization of the unsteady aerodynamic analysis. A previous paper presented the results of the concurrent adaptation of the aeroelastic analysis on a shared memory system with a small number of processors. The proposed paper will present the results of our experience with a distributed memory system having more processors.

Parallel Processing System

In terms of processor interconnections, there are two extreme possibilities. The first is a shared memory architecture where all the processors have access to a common global memory. All inter-processor communication takes place via the common global memory. The second is a distributed memory architecture that generally has a larger number of processors with attached local memories and there is no shared memory. Here, the inter-processor communication takes place through a network of communication links joining selected processors. A hybrid architecture of having some shared memory along with the local memories is also possible but is beyond the ability of the hardware available for the present investigation.

On a shared memory system, implementation of inter-processor communication is trivial to the application programmer because one processor can write data into a location in global memory and another processor can read the data from the same location in memory. While algorithm design is simplified, a penalty is paid in terms of longer memory access times because of the memory contention problem when several processors need to access the shared memory simultaneously.

In a distributed memory architecture, memory contention by the separate processors is absent because each processor has its own local memory. However, inter-processor communication protocol must be explicitly specified by the programmer and excessive delays in communication may degrade the performance of the computational task. In addition, the programmer must decide the manner in which processors are to be connected together, i.e., select a network topology. An implementation of aeroelastic analysis of propfans on a shared memory system was previously presented (Murthy and Janetzke, 1990). In the present paper, the implementation of the aerodynamic analysis portion on a much less expensive distributed memory system is presented. The speedups achieved on

the distributed memory system are compared to those achieved on the shared memory system.

The distributed memory system used in this study is a network of transputer processors. The transputer is a high performance Reduced Instruction Set Computer (RISC) and is made possible by advances in VLSI microcircuit technology that have allowed the placement of an entire computer with memory and communication channels on a single chip. The word transputer was derived from the words 'transistor' and 'computer' suggesting that the processor can be used as an element of a network. One of the chief attractions of a transputer network system is its low cost relative to computers of comparable performance.

The transputer system used in this study consists of one root transputer and 32 network transputers. Each transputer is mounted on a module along with a fixed amount of memory. Each transputer is a IMS T800-20 with a 32-bit central processing unit, 4 kbytes of one cycle (50 ns) on-chip RAM, a floating point computation unit, and four bi-directional serial links. The transputers operate at 20 MHz and are capable of 10 MIPS and 1.5 Mflops. The serial links transmit data at 20 Mbits/sec.

The root transputer module has 8 Mbytes of 5 cycle DRAM and is mounted on an IMS B008 mother board. The B008 board is plugged into a card slot of an IBM PC/AT compatible personal computer which acts as the host to the transputer network. The B008 board contains hardware to interface the root transputer with the PC system.

The network transputer modules each have 1 Mbyte of 3 cycle DRAM. Four network transputer modules are mounted on a B008 board which provides interconnection between modules and power. Eight B008 boards are plugged into the card slots of a PC/AT expansion box. Twisted wire pairs link the root transputer and the network motherboards.

Transputers can be programmed in high-level languages such as C, FORTRAN and Pascal. However, when concurrency is required in a program, OCCAM language (Hoare, 1988) must be used to create parallel processes. OCCAM combines the advantages of a high-level language while allowing efficient, straightforward access to the special hardware features available on the transputer. OCCAM was designed from the start as a programming language to be used in multiprocessor environments.

Several operating systems exist for running programs on transputer systems. Not all of them can handle mixed language programs. Because of its ability to handle compiled FORTRAN programs, an INMOS software system called D705 OCCAM toolset was used. The OCCAM toolset consists of an OCCAM compiler, linker, configurer, file server, and libraries for arithmetic, input/output, and board support functions. Additionally, a FORTRAN compiler by Lattice Logic Ltd.(3L) was used. The 3L FORTRAN compiler provides extra functions to send and receive messages across channels to other concurrently executing processes.

The compilers, the linker and the configurer run on the host PC. The compilers and the linker produce executable modules from the OCCAM and FORTRAN source programs. The configurer is used to generate a loadable file which contains all the executable modules and the network location for each module. This loadable file, called the configured program, is loaded by the file server on the network. The file server also runs on the host PC but concurrently with the network. It also provides keyboard, screen, and magnetic storage disk services for the root transputer.

Since transputer processors have four communication links, it is possible in principle to connect each processor to four other processors. Hence, several possibilities exist in respect to the choice of network topology. Some possible choices are pipeline, tree, ring, grid and torus. In a pipeline topology, all the processors are linked in a one-dimensional chain, as shown in Figure 1. The pipeline topology is the easiest to implement because of its simple inter-processor communication. In this investigation, pipeline topology was used as the primary network topology.

Numerical Procedure

The aeroelastic analysis procedure used in this study was developed at NASA Lewis Research Center for rotating blades (Kaza et al, 1989; Kaza et al 1988). This procedure is implemented in a FORTRAN program called ASTROP3. This program performs flutter and forced response analysis of propfans. The unsteady aerodynamic model neglects thickness effects. In the following, the unsteady aerodynamic model used in ASTROP3 is briefly reviewed to aid in the understanding of the concurrent processing implementation.

The propfan is assumed to have identical groups of blades symmetrically distributed about a rigid disk rotating at a fixed speed Ω in an axial flow of Mach number M. The unsteady aerodynamic forces are calculated by integrating the unsteady pressure disturbance over the blade surface. The unsteady pressure disturbances and the normal velocity over a thin blade are related by an integral equation. Assuming simple harmonic motion with a frequency ω in a three-dimensional potential flow, this integral equation can be written, after appropriate linearization, as (Williams, 1990)

$$W(P) = - B \int_A \Delta\bar{p}(P_0) \frac{\partial}{\partial\bar{\theta}}[K(P,P_0)]\, dA_0 \qquad (1)$$

where P and P_0 represent points on the blade surface, A represents the blade surface, ω is the frequency of blade vibration, B is a constant dependent on flow conditions and K is a kernel function. W and $\Delta\bar{p}$ are proportional to the normal velocity and unsteady pressure disturbance respectively.

Equation (1) was discretized by splitting the blade into n_p quadrilateral panels within each of which $\Delta\bar{p}$ is assumed constant. (See Figure 2). This discretization results in the

algebraic system of equations given by

$$\{W\} = [C]\ \{\Delta\bar{p}\} \qquad \text{Or,} \qquad \{\Delta\bar{p}\} = [C]^{-1}\ \{W\} \tag{2}$$

where $\{W\}$ is a vector of the values of W at chosen control points on each of the panels, $\{\Delta\bar{p}\}$ is a vector of the values of $\Delta\bar{p}$ on each of the panels, and $[C]$ is a matrix of aerodynamic influence coefficients given by

$$c_{ij} = -\int_{A_j} \frac{\partial}{\partial\bar{\theta}}\left[K(P_i,P_0)\right]\, dA_0 \tag{3}$$

where the subscripts i and j refer to the control panel and pressure panel respectively. In terms of radial coordinate r and chordwise azimuthal coordinate $\bar{\theta}$, (Figure 2), eq. (3) can be rewritten as

$$c_{ij} = -\int_{r_{1j}}^{r_{2j}} \int_{\bar{\theta}_{jLE}}^{\bar{\theta}_{jTE}} \frac{\partial}{\partial\bar{\theta}_0}\left[K(\bar{\theta}_i-\bar{\theta}_0, r_i, r_0)\right] r_0\, d\bar{\theta}_0\, dr_0 \tag{4}$$

The subscripts jLE and jTE refer to the leading and trailing edges, and the subscripts $1j$ and $2j$ to the inner and outer edges, of the j -th panel respectively. The definition of the quantities, r_{1j}, r_{2j}, $\bar{\theta}_{jLE}$ and $\bar{\theta}_{jTE}$ is shown in Figure 1. The chordwise integration in eq. (4) can be performed analytically, so that

$$c_{ij} = D_{ijLE} - D_{ijTE} \tag{5}$$

where

$$D_{ijLE/TE} = \int_{r_{1j}}^{r_{2j}} K(\bar{\theta}_i - \bar{\theta}_{0LE/TE}\ ,\ r_i\ ,\ r_0)\ dr_0 \tag{6}$$

Thus, the evaluation of the influence coefficient c_{ij} requires numerical integration of the kernel function K in the radial direction only. The kernel function must itself be evaluated by numerical integration. We refer to r_i as the control panel row radius and r_0 as the pressure panel row radius. Because of the analytical integration in the chordwise direction, the computational effort is nearly independent of the number of chordwise panels and is approximately proportional to the square of the number of radial panel rows.

Once the influence coefficients are evaluated, the generalized motion-dependent force matrix is determined by numerical integration over the blade surface

$$A_{nm} = \sum_{j=1}^{n} \Delta p_{jm}\, \delta_{jn}\, dA_j \tag{7}$$

where Δp_{jm} is the pressure differential across the j-th panel, with area dA_j, due to motion

in the m-th assumed mode shape, and δ_{jn} is the normal displacement of the blade surface at the control point of the j-th panel in the n-th mode shape. The motion-independent aerodynamic force vector is similarly calculated by

$$f_n = \sum_{j=1}^{n} \Delta p_{jF} \delta_{jn} \, dA_j \tag{8}$$

where the subscript F represents the assumed forcing distribution.

The results presented in this paper were obtained using the SR3C-X2 propfan rotor with eight identical blades for flutter analysis. This rotor design was earlier analyzed using ASTROP3 (Kaza et al 1989; Kaza et al 1988). The rotational speed was set at 5280 rpm. The blade surface was discretized such that there were 8 panels in the chordwise direction in each radial row. The number of panel rows in the radial direction was either 9 or 18. Thus the total number of panels was 72 or 144 respectively. The results presented are for the inter-group phase angle at flutter, which was 225° at the given rotational speed. For Mach number and frequency, the values M=0.5 and ω=310 Hz are used.

Computer Implementation

Selection of Parallel Subtasks

The determination of the aerodynamic influence coefficient matrix [C] requires the evaluation of the kernel function and its radial integration (eq. (6)) for all combinations of control panel rows and pressure panel rows on the blade surface. In both flutter and forced response problems, the computation of the kernel function is very expensive as it involves wake integration which requires the numerical evaluation of an integral with an infinite limit. Thus, the kernel function computation and hence the evaluation of the aerodynamic influence coefficients is the dominant contributor to the calculation time required to compute the generalized unsteady aerodynamic forces. For example, in a typical case, the sequential computation of the aerodynamic influence coefficients was found to consume 97.1 percent of the time required for the calculation of the critical Mach number and frequency (see Murthy and Janetzke, 1990).

Fortunately, the computation of the influence coefficients possesses a high degree of independence so that an aeroelastic analysis program using a parallel processing computer would greatly benefit from concurrentization of their computation. Simply put, the kernel function $K(\{\overline{\Delta\theta_i}\},r_i,r_0)$ is evaluated by interpolation after constructing a table of values for K at various values of $\{\overline{\Delta\theta_i}\}$ and fixed values of r_i and r_0. The number of values of r_i and r_0 is given by the number of panel rows used along the radial direction in the discretization of the planform. Once the blade geometry, motion and the flow conditions

are given, the table of values of $K(\{\overline{\Delta\theta_i}\},r_i,r_0)$ at various values of $\{\overline{\Delta\theta_i}\}$ for fixed values of r_i and r_0 can be computed independently of the value of the kernel function at other values of r_i and r_0. Thus, the kernel function $K(\{\overline{\Delta\theta_i}\},r_i,r_0)$ can be evaluated completely in parallel for different values of r_i and r_0. The radial integration of eq. (6) can also be performed in parallel for different values of r_i and r_0. Because of the complete independence of the computations for different values of r_i and r_0, these computations can be performed asynchronously.

Communication Software

The computation of the influence coefficients for given values of pressure and control panel radii is selected as an independent subtask. This task is performed by a subtask FORTRAN program which runs on the network processors. A main FORTRAN program executing on the root processor handles the computation required for setting up the panel discretization. A supporting OCCAM program, described below, executes concurrently with this main FORTRAN program and handles the input/output operations and directs the initiation and shutdown of network computations.

With the INMOS Standalone D705 OCCAM toolset software system, mentioned previously, concurrent operation of a FORTRAN program on a transputer system requires an OCCAM program called a harness. An OCCAM harness program enables the FORTRAN program to internally send and receive data with other processes operating on the same transputer and running in parallel. An OCCAM harness also provides the means for a transputer to send and receive data through its four external links with other transputers in the network and with the host computer.

Two OCCAM harness programs were written to implement the program on the transputer network. One harness program resides on the root transputer and the other on each of the network transputers. The root harness program provides data transfer for input/output functions in the main FORTRAN program to the keyboard, screen, and magnetic disks of the host PC. It also directs data to and from the network links. Similarly, the network harness program passes data to and from the FORTRAN subtask program and other network processors. An abridged listing of the OCCAM and FORTRAN source codes is given in the Appendix.

The root OCCAM harness program, after some routine setup operations, has three processes running in parallel: the main FORTRAN program, a network data handler and an i/o handler. The main FORTRAN program prepares the required data and calls special 3L FORTRAN routines (CHANOUTMESSAGE and CHANINMESSAGE) that send data to and receive data from the network data handler process. The i/o handler is a standard INMOS toolset process which multiplexes read/write requests from the main FORTRAN program to a standard file server program which runs concurrently on the host PC. The

network handler process receives global input data from the main FORTRAN process and sends it to the network processes. After sending the global input data, the network handler process executes two relay processes in parallel. One of these processes relays requests from the main FORTRAN process to the network for specific transputers to compute the coefficients for a specific pressure/control panel row ordered pair. The other process relays the computed influence coefficients from the network to the main FORTRAN process.

The network OCCAM harness program initially does some standard setup and then has two processes running in parallel: the subtask FORTRAN process and a network handler. The subtask FORTRAN process receives the global input data sent from the main FORTRAN process and then receives the requests for computation of specific influence coefficients. It sends back the results and then waits for the next request for more computation or shutdown. The network handler process in the network harness is similar to that in the root harness. It initially receives the global input data from the root (generally via a neighboring network transputer) and passes it to the other neighboring transputer, if it exists, and also to the subtask FORTRAN process. After relaying the global input data, the network handler executes two relay processes in parallel. One of these processes relays requests from the root to the rest of the network and also to the subtask FORTRAN process if appropriate. The other process relays results in the opposite direction, i.e., from either the subtask FORTRAN process or the rest of the network to the root.

In addition to the two harness programs, another OCCAM program, called a configuration program, is also used. The configuration program places the root and network processes on the root and the network transputers, as appropriate. It also assigns physical transputer links to process communication channels. It is this program that defines the topology of the network.

Note that the configuration program and the network handler programs are topology-dependent and would need to be significantly altered for network topologies other than a pipeline.

Processor Idle Time and Its Reduction

Processor idle time is the time a processor is not doing any computation while at least one other processor is. If processors are fully utilized, processor idle time will be zero. This ideal condition exists when the parallel subtasks are known to be of equal computational time and their number is an integral multiple of the number of processors used. In these circumstances, the subtasks could be allocated among the available processors in equal numbers and no processor would be left idle during the computational task. This is simple to implement because network communications are required only at the initiation and the termination of the computation.

In general, however, the number of subtasks is not an integral multiple of the

number of processors. Also, the computation times for subtasks generally are neither equal nor known *a priori*. Any one of these conditions usually results in some processors being idle for a significant part of the computation time. For example, in the calculation of the aerodynamic influence coefficients, the computation time for coincident control and pressure panel rows is approximately twice that otherwise (Murthy and Janetzke, 1990). Also, these computation times, being dependent on the flow conditions and blade geometry, cannot be known *a priori*. Consequently, equal allocation of subtasks, as described above, results in significant processor idle time or under-utilization of the processors.

In order to obtain better processor utilization in the present investigation, the subtasks were sequentially allocated to processors as they became available for computation. In such an asynchronous operation, the processors become idle only towards the end of the parallel computational phase when no subtasks remain for execution.

Significant processor idle time is still possible if a combination of subtasks, having large differences in computational times among them, is executing near the end of the parallel computational phase. This occurs, for example, if the standard order of looping over the control panel rows and pressure panel rows from hub to tip is used because the last subtask has coincident control and pressure panel rows. By appropriately ordering the sequence of computation of the parallel subtasks, this processor idle time is reduced. For one such ordering, all subtasks having coincident pressure and control panel rows are started before those having non-coincident pressure and control panel rows. This ordering scheme is referred to as the diagonal ordering scheme. A more detailed discussion of subtask ordering is given by Murthy and Janetzke (1990). Unless otherwise mentioned, all the results reported were obtained by the diagonal ordering scheme.

Results

For all the results reported in this paper, the root processor and one or more network processors are used. The root processor is used for file input and screen output and for directing the network operation. The processing time on the root processor for non-i/o operations is negligible and no i/o operations are included in the measurement of calculation times reported in this paper. In this report, speedup is the ratio of the computational time using one network processor to that using one or more network processors. Efficiency is defined as the speedup divided by the number of processors. Speedup is a measure of the reduction in the effective calculation time achieved by the parallel algorithm whereas efficiency is a measure of the processor utilization.

Figure 3 shows the total time for the computation of unsteady aerodynamic influence coefficients as a function of the number of network processors used. The results are for 9 radial panel rows. The total time decreases rapidly as the number of processors is increased from one. As the number of processors is increased, the total time decreases much less rapidly. Hence, it requires only a few processors to effect a large reduction in

the computational time compared to the sequential processing time. This is a direct consequence of the natural parallelism in the algorithm and the near-linear speedup achieved.

Figure 4 and Figure 5 respectively show the speedup and processor efficiency achieved as a function of the number of processors. The solid lines indicate the ideal conditions of linear speedup and 100 percent processor efficiency. The speedup improves and efficiency generally degrades as more processors are used. Speedup improves because the computation is divided among more and more processors. The efficiency degrades because more processors become idle towards the end of the computation and increased communication times result as more processors are used. It is presumed that communication time is of lesser significance because the it is much smaller than the computation time of a subtask for this application. However, the relative importance of these two factors has not been investigated at the present time.

Also noted in Figure 4 and Figure 5, the speedup and efficiency improve in jumps at certain points. These jumps are most noticeable when the number of processors are 23 and 30. These jumps occur when the subtasks become better balanced among the processors as one more processor is added to the network. The points at which the jumps occur depend on the number, execution order and the execution times of the subtasks.

Effect of Subtask Ordering

Changing the ordering of subtasks from the standard scheme to the diagonal scheme results in a reduction in the processor idle time. For a given problem, this reduction is nearly independent of the number of processors used. When the number of processors used is small, the total computation time is large and the percent reduction in total time is small. As the total computation time decreases due to a larger number of processors, the percent reduction generally becomes larger. Thus the beneficial effect of diagonal subtask ordering is significant only for a large number of processors. The influence of the ordering scheme on the speedup is illustrated in Figure 6. Due to the reduction in processor idle time, the diagonal ordering scheme brings the speedup closer to linear speedup. The processor efficiency, although not shown, is also improved significantly as the number of processors used becomes larger.

Effect of Problem size

To study the effect of problem size, the calculations were repeated for 18 radial row panels. This doubles the number of panels and thus quadruples the number of parallel subtasks to be executed. The variation of the total time for 18 and 9 radial row panels is shown in Figure 7. The results are qualitatively similar. The additional computational time due to a bigger problem size decreases as the number of processors is increased. A comparison of the variation of speedup for 9 and 18 radial panel rows is shown in Figure 8. The jumps in speedup, earlier referred to in the discussion of Figure 4, are less severe for

the bigger problem. The speedup is slightly higher for the bigger problem. However, this is not necessarily a general trend.

Effect of Network Topology

To study the effect of network topology, the calculations were repeated after replacing the pipeline with a binary tree. The binary tree topology used is shown in Figure 9. Since only 32 network transputers were available, a four-level binary tree topology, which uses 30 transputers, was implemented. At the tree root, the root transputer is connected to two branch transputers. Note that the binary tree topology uses three of the four communication links on each branch transputer, whereas the pipeline topology uses only two. As a result, the average number of steps for processor communication to the root is much less for the binary tree. It was found that the effect of changing the network topology on speedup is very small for the case of 9 radial panel rows. For the bigger problem of 18 radial panel rows, the effect was significant and is shown in Figure 10. For this case, the speedup variation was closer to linear speedup with the tree topology than with the pipeline topology.

Shared Memory vs. Distributed Memory

The present results using a distributed memory computer are compared to those using a shared memory computer. The shared memory implementation is described in Murthy and Janetzke (1990). The shared memory computer used, an Alliant FX/80, has a maximum of eight processors. The speedup obtained on both these computers is shown in Figure 11 as a function of the number of processors. It is clear that the distributed memory implementation is capable of achieving a speedup closer to the ideal of linear speedup, indicating that, for the present algorithm, memory contention imposes a more severe penalty than inter-processor communication. However, the processors on the FX/80 are significantly faster than the transputers used in the distributed memory computer. The actual time of computation on the transputer system with 32 processors is comparable to that on the FX/80 using eight processors.

Conclusions

The interest in this study lies in the demonstration of a relatively inexpensive transputer-based parallel processing system as a viable alternative to shared memory systems for practical aeroelastic analysis. The demonstration was through an actual implementation of distributed memory parallel computation of the unsteady aerodynamic analysis portion required in typical aeroelasticity applications, using two network topologies. It was found that speedups close to linear speedup can be achieved for this portion of analysis, indicating the overhead resulting from parallelization is low. The actual speedup achieved depends on the number of processors, the scheduling algorithm, the problem size and the network topology. The effects of all these factors has been examined.

For distributed memory systems, where the number of processors is typically large, the dynamic scheduling of the independent subtasks plays an important role in achieving high speedup. Efficiencies up to 86 percent using 32 processors are demonstrated.

When the problem size is increased by quadrupling the number of parallel subtasks, the speedup remained high. While this is not a general observation, it indicates that bigger problems would benefit more from parallelization.

Also, speedup is virtually unaffected by the network topology except when the problem size is big. This indicates that the simplest network topology, in which the inter-processor communication is easy to manage, can be chosen without a substantial penalty.

One limitation of the present investigation is that conclusions drawn may not necessarily be applicable to aeroelastic analysis, since only a potion of the complete analysis has been investigated. However, this is a portion involving a large computational effort. Thus the conclusions are preliminary in nature and the significant question remains: what is the potential of the present implementation to produce high speedups in the context of a complete aeroelastic analysis ? An investigation of this question is in progress.

References

Hoare, C. A. R., 1988, OCCAM 2 Reference Manual, Prentice Hall International Series, Prentice Hall, New York.

Kaza, K. R. V., Mehmed, O., Narayanan, G. V. and Murthy, D. V., 1989, "Analytical Flutter Investigation of a Composite Propfan Model", Journal of Aircraft, Vol. 26, No. 8, pp. 772-780.

Kaza, K. R. V., Williams, M. H., Mehmed, O. and Narayanan, G. V., 1988, "Aeroelastic Response of Metallic and Composite Propfan Models in Yawed Flow", NASA TM-100964.

Murthy, D. V. and Janetzke, D. C., 1990, "Concurrent Processing Adaptation of Aeroelastic Analysis of Propfans", presented at the AIAA/ASME/ASCE/AHS/ASC 31st Structures, Structural Dynamics and Materials Conference, Long Beach, CA (AIAA-90-1036-CP).

Noor, A. K. and Venneri, S., 1990, "Advances and Trends in Computational Structural Mechanics", Computer Systems in Engineering, Vol. 1, No. 1, pp.23-36.

Noor, A. K. and Atluri, S. N., 1987, "Advances and Trends in Computational Structural Mechanics", AIAA Journal, Vol. 25, No. 7, pp. 977-995.

Williams, M. H., 1990, "An Unsteady Lifting Surface Method for Single Rotation Propellers", NASA CR-4302.

Appendix

The listings given in this section are presented to show how the sequential FORTRAN program was modified for parallel execution on a pipeline transputer network. Also given are the specially written OCCAM programs which provide the interface for link communications on the transputer network. Although a knowledge of OCCAM and FORTRAN programming is necessary to fully understand the procedures, the listings are fairly readable and comments are interspersed throughout to help the reader.

The first listing is that of the configuration program. This program, written in OCCAM, defines the network topology. It specifies which communication links are being used and how they are interconnected. It assigns the executable processes and the communication channels required to specific transputers on the network.

The second set of listings encompasses all the processes that run on the root transputer. The main OCCAM harness program has three processes which are active concurrently: a root transputer-to-PC communication interface, a root-to-network transputer interface, and a FORTRAN main program. The root/PC interface process, called io.handler, is a standard INMOS Toolset process and not listed herein. It multiplexes read/write data for the FORTRAN main program between the root transputer and the host PC. The root/network interface process, called network.handler, relays data for the FORTRAN main program between the root transputer and the first network transputer. The details of the network.handler process are given within the OCCAM listing. The main FORTRAN process, called PROGRAM aeroco, is shown in an abridged listing which shows the flow of the program and the special subroutine calls for data communication with the OCCAM network.handler process. The data communication subroutines used, CHANOUTMESSAGE and CHANINMESSAGE, are special routines supplied by the 3L FORTRAN runtime library. The three arguments for these subroutines are: a channel index (always 2), variable name, and message size in number of bytes.

The third set of listings encompasses all the processes that run on each of the network transputers. The network OCCAM harness program has only two concurrent processes: a transputer-to-transputer interface and a FORTRAN subtask program. The interface process, also called network.handler as on the root transputer, similarly relays data for the network and the local FORTRAN subtask program. The listing of the network FORTRAN is abridged to show only the flow of the program and the special data communication subroutine calls.

Configuration program

```
#SC "acoef.lsc"       -- executable on the root processor
#SC "inflpr.lsc"      -- executable on each network processor

VAL B008quads IS 8 :
VAL no.of.netw.prcrs IS 4*B008quads :
VAL no.of.t8s IS no.of.netw.prcrs+1 :
VAL root IS 0 :

--{  channel addresses
--   for the 4 communication links on any transputer
VAL link0.in  IS 4 :
VAL link1.in  IS 5 :
VAL link2.in  IS 6 :
VAL link3.in  IS 7 :

VAL link0.out IS 0 :
VAL link1.out IS 1 :
VAL link2.out IS 2 :
VAL link3.out IS 3 :
--}

--{  CHAN definitions
[no.of.t8s]CHAN OF ANY to, from :
CHAN OF ANY to.server, from.server :
--}

-- Define pipeline architecture
PLACED PAR
  PLACED PAR

    -- Root processor
    PROCESSOR root T8
      PLACE from.server  AT  link0.in  :
      PLACE to.server    AT  link0.out :
      PLACE from[0]      AT  link2.out :
      PLACE to[0]        AT  link2.in  :
      acoef(from.server, to.server,
            from[0], to[0], root, no.of.netw.prcrs)

  --  Network B008quad pipeline
  PLACED PAR i = 1 FOR no.of.netw.prcrs
    PLACED PAR
      -- Network processor (i)
      PROCESSOR i T8
        PLACE from[i-1]  AT  link1.in  :
        PLACE to[i-1]    AT  link1.out :
        PLACE from[i]    AT  link2.out :
        PLACE to[i]      AT  link2.in  :
        influc.process(from[i-1], to[i-1], from[i], to[i], i)
```

Root processor program

```
                    Root OCCAM harness program

PROC acoef(CHAN OF ANY from.server, to.server,
                       to.network, from.network,
                VAL INT process.id, VAL INT no.of.netw.prcrs)

  #USE "c:\toolset\realio"     -- occam i/o primitives library
  #USE "c:\toolset\flibs"      -- File server library
  #USE "c:\toolset\boardlib"   -- Board support library

  #IMPORT "acoef.main (INT dummy, VAL INT flag, []INT ws1, in,
                         out, ws2) = 1"     -- FORTRAN main program

  --  Network handler process for root
  PROC network.handler (CHAN OF ANY from.main, to.main,
                                    from.network, to.network)

    --pass data from main process to network, and vice versa
    [208]INT buffer :
    [3]INT request :
    INT array.size :  -- number of return values
    BOOL running, runnin2 :
    CHAN OF BOOL done :
    VAL INT input IS 207 :  -- number of global input values

    SEQ
      -- send number of processors available
      to.main ! no.of.netw.prcrs

      -- read global input data
      from.main ? buffer[0]
      from.main ? [buffer FROM 1 FOR 33]
      from.main ? [buffer FROM 34 FOR 1]
      from.main ? [buffer FROM 35 FOR 20]
      from.main ? [buffer FROM 55 FOR 20]
      from.main ? [buffer FROM 75 FOR 20]
      from.main ? [buffer FROM 95 FOR 20]
      from.main ? [buffer FROM 115 FOR 20]
      from.main ? [buffer FROM 135 FOR 20]
      from.main ? [buffer FROM 155 FOR 20]
      from.main ? [buffer FROM 175 FOR 12]
      from.main ? [buffer FROM 187 FOR 20]

      -- send global input data to network
      to.network ! [buffer FROM 0 FOR input]
      array.size := (2*buffer[5])*(buffer[11]*buffer[11])
      --  array.size := 2*ND*NXP*NXP
```

```
    running := TRUE
    runnin2 := TRUE
    PAR
      WHILE running    -- relay requests from main program
        SEQ
          from.main ? [request FROM 0 FOR 3]
          to.network ! [request FROM 0 FOR 3]
          IF
            request[0] <= 0
              SEQ
                running := FALSE
                done ! running
            TRUE
              SKIP
      WHILE runnin2    -- relay results to main program
        ALT
          from.network ? [buffer FROM 0 FOR array.size+3]
            SEQ
              to.main ! buffer[0]
              to.main ! buffer[1]
              to.main ! buffer[2]
              to.main ! [buffer FROM 3 FOR array.size]
          done ? runnin2
            SKIP
    from.main ? buffer[0]   -- all normal output finished
:   -- end of network handler process

VAL flag IS 1:  -- use combined work space

[3]INT out.pointer, in.pointer :
[2]CHAN OF ANY reserved.out, reserved.in :
CHAN OF ANY from.acoef.main, to.acoef.main :
INT d :
[200000]INT ws1 :
[1]INT dummy.ws :

SEQ
  -- connect pointers from FORTRAN program
  --                      to occam I/O channels
  SEQ i = 0 FOR 2
    SEQ
      LOAD.OUTPUT.CHANNEL(out.pointer[i], reserved.out[i])
      LOAD.INPUT.CHANNEL(in.pointer[i], reserved.in[i])
  LOAD.OUTPUT.CHANNEL(out.pointer[2], from.acoef.main)
  LOAD.INPUT.CHANNEL(in.pointer[2], to.acoef.main)

  SEQ
    PAR
      io.handler(from.server, to.server,
                 reserved.out, reserved.in)
```

```
        network.handler(from.acoef.main, to.acoef.main,
                                from.network, to.network)

        acoef.main(d, flag, ws1,
                        in.pointer, out.pointer, dummy.ws)

      terminate.filer(from.server, to.server, result)
:


                    Main FORTRAN Process

      PROGRAM acoef.main
          :
          :  declaration statements
          :
C      GET NUMBER OF PROCESSORS AVAILABLE ON NETWORK
      CALL CHANINMESSAGE(2, navprs, 4)
 10   READ (5,*) IJOB, NPROCS, IOPT
      IF (NPROCS .GT. NAVPRS) THEN
        NPROCS = NAVPRS
        WRITE(6,12) NPROCS
 12     FORMAT(' NPROCS INPUT EXCEEDS THE AVAILABLE NUMBER OF',
     X    ' PROCESSORS'/'  NPROCS RESET TO ',I3)
       ENDIF
C      LET NETWORK KNOW HOW MANY PROCESSORS ARE TO BE USED
      CALL CHANOUTMESSAGE(2,NPROCS,4)

      READ(5,100) TITLE
         :
          :  read input data
          :
      CALL FLUTAT(NPMX)
     CALL CHANOUTMESSAGE(2,NPMX,4)
       print*,' message sent to indicate end of main program '
      STOP
      END

      SUBROUTINE FLUTAT(NPMX)
         :
          : declaration and initialization statements
          :
       CALL PRPANT (IPRNT,NBLD,MAX,S,NRT0,RT,XT,THTB,XL,
     x                    THLB,NXP0,TSE,PMX,NMODE,NMDMX,NPH,IPH,
     x                    NOM,OM,CAC,WA,CQ)

      RETURN
      END

      SUBROUTINE PRPANT (IPRNT,NBLD,MAX,S,NRT0,RT,XT,THTB,XL,
     x                    THLB,NXP0,TSE,PMX,NMODE,NMDMX,NPH,IPH,
```

```
     x                    NOM,OM,CAC,WA,CQ)
          :
          : declaration and initialization statements
          :
C               COMPUTE INFLUENCE COEFFICIENT MATRICES
      CALL INFLUC(NPMX,IPH,RT,SIGT,ALF,THT,THL,CHD,TSE,M0T,CAC)
      RETURN
      END

      SUBROUTINE INFLUC(NPMX,IPH,RT,SIGT,ALF,
     x                  THT,THL,CHD,TSE,M0T,CAC)
C
C     Special routine for interfacing with OCCAM process
C     This routine sends out requests for computation on the
C     network andd then receives and stores the results.
C
      IMPLICIT REAL(A-H,L-M,O-Z)
      LOGICAL RUNNIN
      DIMENSION  RT(1),SIGT(1),ALF(1),M0T(1),THT(1),
     x           CHD(1),TSE(1),THL(1),IPH(1)
      DIMENSION  IREQST(3)
      COMPLEX  CBUF(144), CAC(NPMX,NPMX,1)
      COMMON/time/NPROCS,IOPT
      COMMON /COMP/ MX,MX2,NB,MT,ND,OMB,
     x ALPH,ALPH0,DSIG,EP,NXP,NRP,IRP1(20),IBPC
C
C       SEND OUT COMMON/COMP/ VALUES
      CALL CHANOUTMESSAGE(2,MX,132)
C       SEND OUT CALL ARGUMENTS DATA
      CALL CHANOUTMESSAGE(2,NPMX,4)
      CALL CHANOUTMESSAGE(2,IPH,80)
      CALL CHANOUTMESSAGE(2,RT,80)
      CALL CHANOUTMESSAGE(2,SIGT,80)
      CALL CHANOUTMESSAGE(2,ALF,80)
      CALL CHANOUTMESSAGE(2,THT,80)
      CALL CHANOUTMESSAGE(2,THL,80)
      CALL CHANOUTMESSAGE(2,CHD,80)
      CALL CHANOUTMESSAGE(2,TSE,48)
      CALL CHANOUTMESSAGE(2,M0T,80)
      NUMBYT = 8*NXP*NXP*ND
      ICPMAX = NRP*NRP
      IF (NPROCS.GT.ICPMAX) NPROCS = ICPMAX
C  Send requests to all processors being used
      DO K = 1,NPROCS
       IPRCS = K
       CALL INDICE(IPRCS,NRP,IRC,IRP,IOPT)
       IDPRCR = NPROCS - K + 1
       IREQST(1) = IDPRCR
       IREQST(2) = IRC
       IREQST(3) = IRP
       CALL CHANOUTMESSAGE(2,IREQST,12)
```

```
        END DO
       ICP = 1
       RUNNIN = .TRUE.
       DO WHILE (RUNNIN)
C  Receive results from a processor on the network
        CALL CHANINMESSAGE(2, IDPRCR, 4)
        CALL CHANINMESSAGE(2, IRC, 4)
        CALL CHANINMESSAGE(2, IRP, 4)
        CALL CHANINMESSAGE(2, CBUF, NUMBYT)
C  Store results
        I = 0
        II0 = (IRC-1)*NXP
        JJ0 = (IRP-1)*NXP
         DO 10 IXC=1,NXP
          II=IXC+II0
          DO 10 IXP=1,NXP
           I = I + 1
           JJ=IXP+JJ0
           CAC(II,JJ,1)=CBUF(I)
   10      CONTINUE
C  Send out next request for that processor
        IF (IPRCS.LT.ICPMAX) THEN
          IPRCS = IPRCS+1
          CALL INDICE(IPRCS,NRP,IRC,IRP,IOPT)
          IREQST(1) = IDPRCR
          IREQST(2) = IRC
          IREQST(3) = IRP
          CALL CHANOUTMESSAGE(2,IREQST,12)
          END IF
        ICP = ICP+1
        IF (ICP.GT.ICPMAX) RUNNIN = .FALSE.
        END DO
C  Send signal to shutdown all processes
       IREQST(1) = -1
       CALL CHANOUTMESSAGE(2,IREQST,12)
       print*,' message sent to shut down network '
       RETURN
       END

       SUBROUTINE INDICE(IRCP,NRP,IRC,IRP,IOPT)
        GO TO (1,2,3),IOPT
C        Standard order: IRC incremented after cycle of IRP
   1    irc = (ircp-1)/nrp + 1
        irp = ircp - (irc-1)*nrp
        RETURN
           :
           :  other ordering option
           :
C        Diagonal order: Main diagonal done first
   3    ircp1 = (ircp-1)*(NRP+1)
        ircpm = MOD(ircp1,NRP*NRP) + 1
```

```
      irc = (ircpm-1)/nrp + 1
      irp = ircpm - (irc-1)*nrp
      RETURN
    END
```

Network processor program

```
                Network OCCAM Harness Program

PROC influc.process(CHAN OF ANY from.host, to.host,
                                to.network,from.network,
                    VAL INT process.id)

  #IMPORT "influc.routine(INT dummy, VAL INT flag,[]INT ws1, in,
                         out, ws2) = 1"  -- Network FORTRAN Subtask

  -- Network network handler process
  PROC network.handler (CHAN OF ANY from.host, to.host,
                                    from.process, to.process,
                                    from.pipe, to.pipe,
                               VAL INT process.id)
    --pass data from host to local process and pipeline,
    -- and vice versa

    [208]INT buffer :
    [3]INT request :
    INT array.size :  -- number of return values
    BOOL pipe.segment, end.of.pipe, running, runnin2 :
    CHAN OF BOOL done :
    VAL INT input IS 207 :  -- number of global input values
    SEQ
      -- read global input data
      from.host ? [buffer FROM 0 FOR input]
      INT number.of.processors IS buffer[0] :
      IF
        process.id = number.of.processors
          SEQ
            pipe.segment := FALSE
            end.of.pipe := TRUE
        TRUE
          SEQ
            pipe.segment := TRUE
            end.of.pipe := FALSE
      IF
        pipe.segment
          SEQ   -- send global input to next processor
            to.pipe ! [buffer FROM 0 FOR input]
        TRUE
          SKIP
```

```
-- send global input data to local INFLUC routine
to.process ! process.id
to.process ! buffer[0]
to.process ! [buffer FROM 1 FOR 33]
to.process ! [buffer FROM 34 FOR 1]
to.process ! [buffer FROM 35 FOR 20]
to.process ! [buffer FROM 55 FOR 20]
to.process ! [buffer FROM 75 FOR 20]
to.process ! [buffer FROM 95 FOR 20]
to.process ! [buffer FROM 115 FOR 20]
to.process ! [buffer FROM 135 FOR 20]
to.process ! [buffer FROM 155 FOR 20]
to.process ! [buffer FROM 175 FOR 12]
to.process ! [buffer FROM 187 FOR 20]
array.size := (2*buffer[5])*(buffer[11]*buffer[11])
--              2*ND*NXP*NXP

running := TRUE
runnin2 := TRUE
PAR
  WHILE running
    -- relay input data from root processor
    SEQ
      -- receive computation request from host processor
      from.host ? [request FROM 0 FOR 3]
      IF
        pipe.segment
          -- relay request to next processor
          to.pipe ! [request FROM 0 FOR 3]
        TRUE
          SKIP
      IF
        request[0] = process.id
          -- send request to local INFLUC routine
          SEQ
            to.process ! request[1]
            to.process ! request[2]
        request[0] <= 0
          -- relay shutdown signal to local processes
          SEQ
            to.process ! request[0]
            running := FALSE
            done ! running
        TRUE
          SKIP
  WHILE runnin2
    -- relay results to root processor
    ALT
      pipe.segment & from.pipe ? [buffer FROM 0 FOR
                                          array.size+3]
        SEQ
```

```
            -- relay results to root processor from network
            to.host ! [buffer FROM 0 FOR array.size+3]
          from.process ? buffer[0]
            SEQ
              -- relay results from local routine
              from.process ? buffer[1]
              from.process ? buffer[2]
              from.process ? [buffer FROM 3 FOR array.size]
              to.host ! [buffer FROM 0 FOR array.size+3]
          done ? runnin2
            SKIP
:

[3]INT out.pointer, in.pointer :
[2]CHAN OF ANY reserved.out, reserved.in :
CHAN OF ANY from.influc, to.influc :
INT dummy :
VAL INT flag IS 1 :  -- for combined work space
[20000]INT ws1 :
[1]INT ws2 :

SEQ
  -- connect pointers from FORTRAN program
  --                    to occam I/O channels
  LOAD.OUTPUT.CHANNEL(out.pointer[2], from.influc)
  LOAD.INPUT.CHANNEL(in.pointer[2], to.influc)
  PRI PAR

    network.handler(from.host ,to.host,
                    from.influc, to.influc,
                    from.network, to.network,
                                  process.id)

    influc.routine(dummy, flag, ws1,
                     in.pointer, out.pointer, ws2)
:
```

Network FORTRAN Process

```
      PROGRAM influc.routine
C     Special routine to relay data with OCCAM harness progam
C     and call subroutine INFLUC
      IMPLICIT REAL(A-H,L-M,O-Z)
      LOGICAL RUNNIN
      DIMENSION  RT(20),SIGT(20),ALF(20),MOT(20),THT(20),CHD(20)
     C  ,TSE(12),THL(20),IPH(8)
      COMPLEX  CAC(144)
      COMMON/COMP/MX,MX2,NB,MT,ND,OMB,ALPH,ALPH0,DSIG,EP,
     C NXP,NRP,IRP1(20),IBPC
```

```
C       read in this processor ID
        CALL CHANINMESSAGE(2,IDPRCR,4)
C       read in how many processors are to be used
        CALL CHANINMESSAGE(2,NPRCSR,4)
C       READ IN COMMON/COMP/ DATA
        CALL CHANINMESSAGE(2,MX,132)
C       READ IN CALL ARGUMENTS DATA
        CALL CHANINMESSAGE(2,NPMX,4)
        CALL CHANINMESSAGE(2,IPH,80)
        CALL CHANINMESSAGE(2,RT,80)
        CALL CHANINMESSAGE(2,SIGT,80)
        CALL CHANINMESSAGE(2,ALF,80)
        CALL CHANINMESSAGE(2,THT,80)
        CALL CHANINMESSAGE(2,THL,80)
        CALL CHANINMESSAGE(2,CHD,80)
        CALL CHANINMESSAGE(2,TSE,48)
        CALL CHANINMESSAGE(2,M0T,80)
C
        NUMBYT = 8*NXP*NXP*ND
C       RESET DIMENSION PARAMETER FOR CAC
        NPMX = NXP
        RUNNIN = .TRUE.
        DO WHILE (RUNNIN)
         CALL CHANINMESSAGE(2,IRC,4)
         IF (IRC.GT.0) THEN
           CALL CHANINMESSAGE(2,IRP,4)
C                COMPUTE INFLUENCE COEFFICIENT MATRICES
           CALL INFLUC(IRC,IRP,NPMX,IPH,RT,SIGT,ALF,
      X                          THT,THL,CHD,TSE,M0T,CAC)
C       SEND OUT COEFFICIENT DATA
           CALL CHANOUTMESSAGE(2, IDPRCR, 4)
           CALL CHANOUTMESSAGE(2, IRC, 4)
           CALL CHANOUTMESSAGE(2, IRP, 4)
           CALL CHANOUTMESSAGE(2, CAC, NUMBYT)
          ELSE
           RUNNIN = .FALSE.
           STOP
          ENDIF
         END DO
        STOP
        END

        SUBROUTINE INFLUC (IRC,IRP, NPMX,IPH,RT,SIGT,ALF,
      C                    THT,THL,CHD,TSE,M0T,CAC)
             :
             :   declaration statements
             :
C    ********** SET CONTROL PT. ROW
CCCC   DO 101 IRC=1,NRP
             :
             :   a few variables dependent only on IRC
```

```
          :
C    **********  SET PANEL ROW
CCCC   DO 101 IRP=1,NRP
         JRP=IRP1(IRP)
           :
           :   computation of a set of influence coefficients
           :
 101   CONTINUE
       RETURN
       END
```

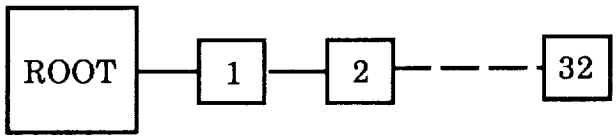

Figure 1. - Pipeline topology of transputer network.

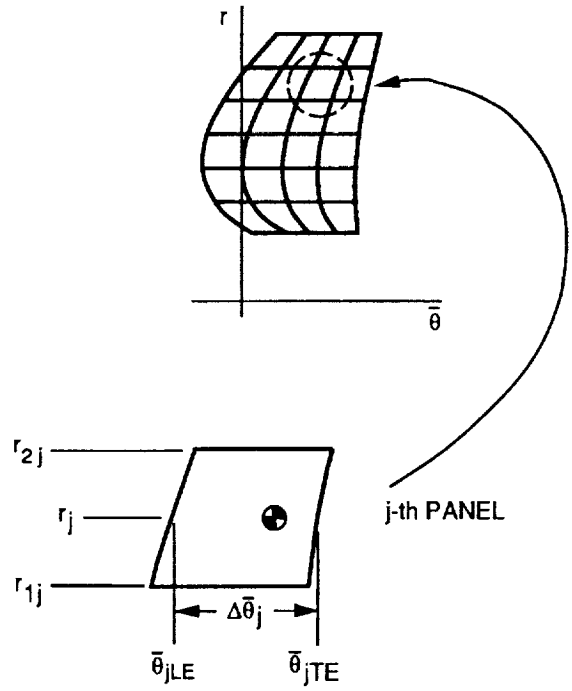

Figure 2. - Blade paneling

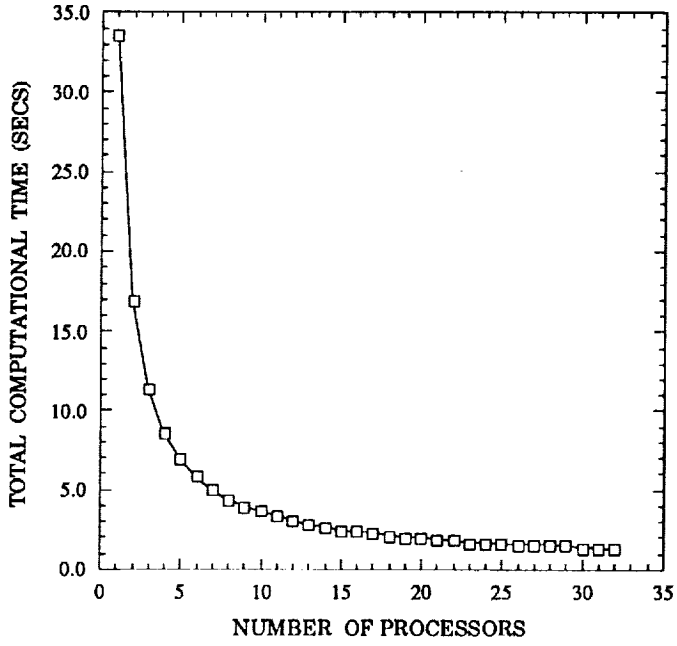

Figure 3. - Total time as a function of number of processors.

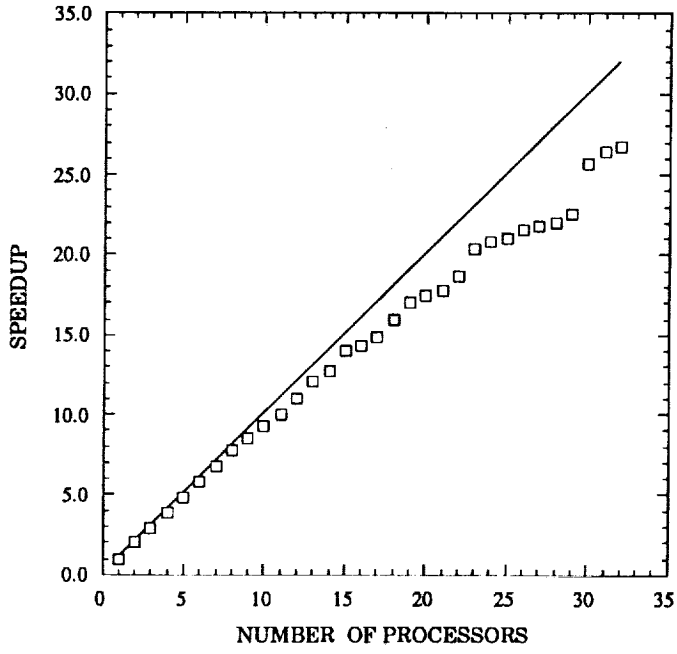

Figure 4. - Speedup as a function of number of processors.

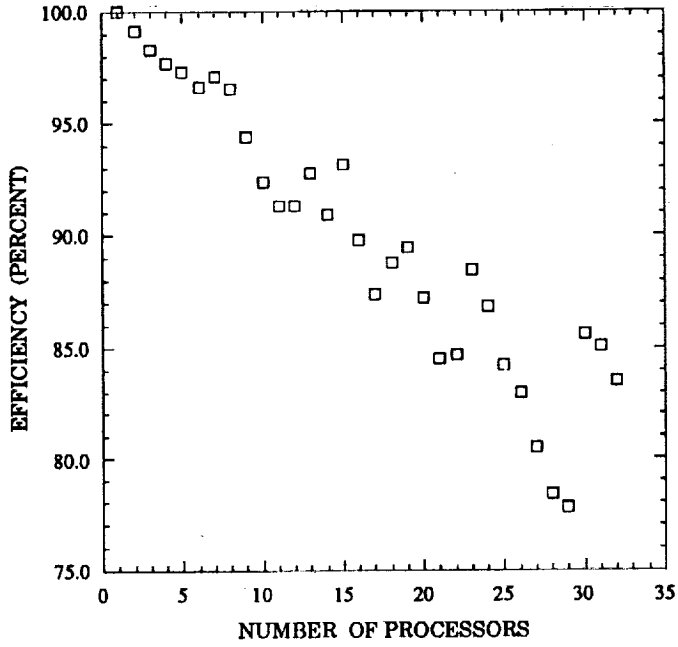

Figure 5. - Processor efficiency as a function of number of processors.

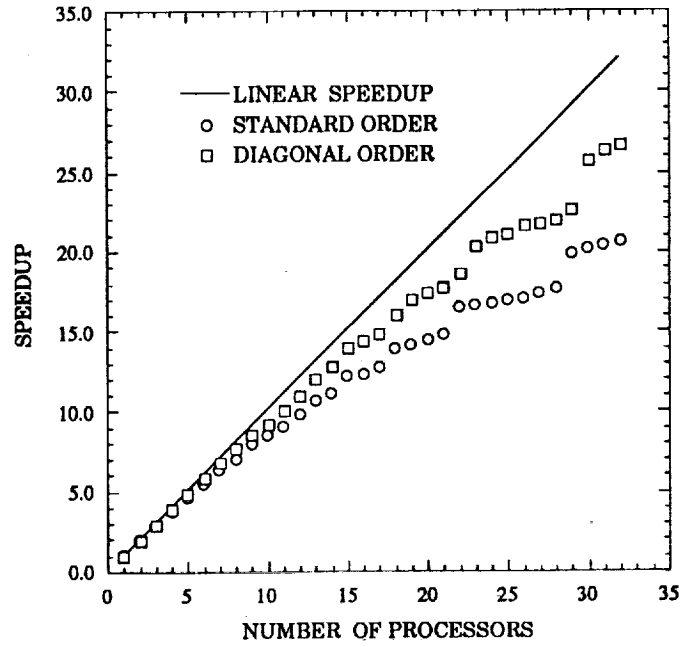

Figure 6. - Effect of subtask ordering on speedup.

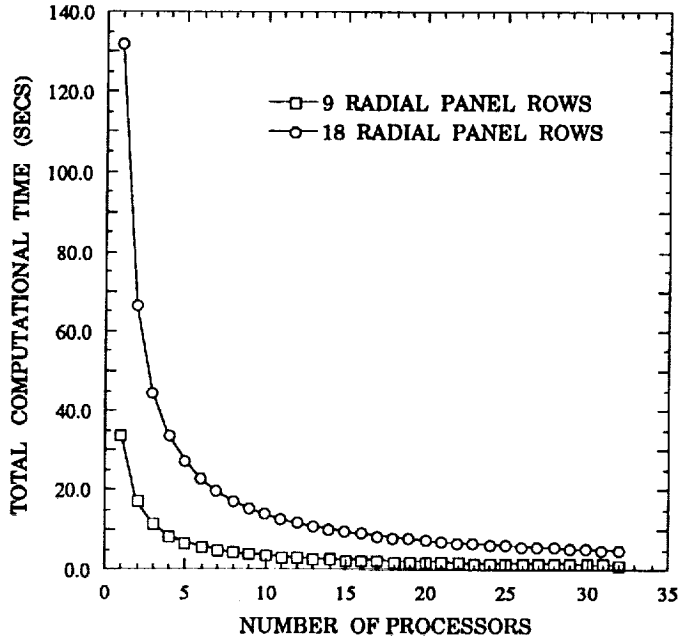

Figure 7. - Effect of problem size on total computation time.

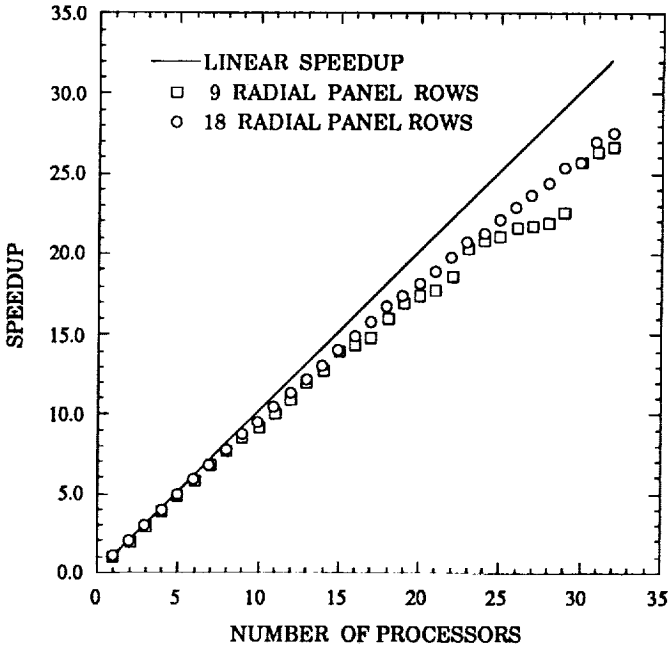

Figure 8. - Effect of problem size on speedup.

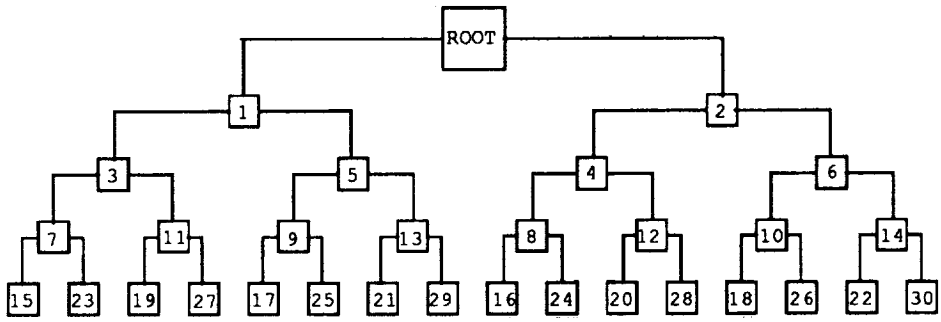

Figure 9. - Binary tree network topology.

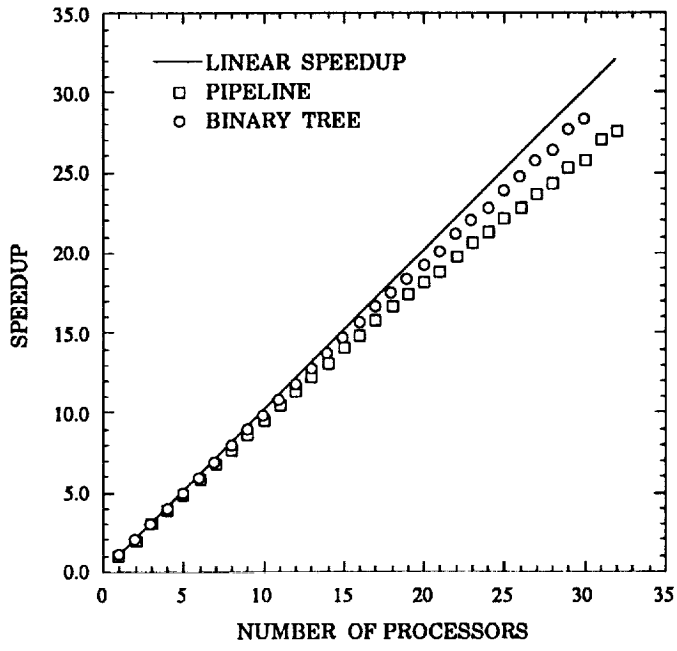

Figure 10. - Effect of network topology on speedup
(18 radial panel rows)

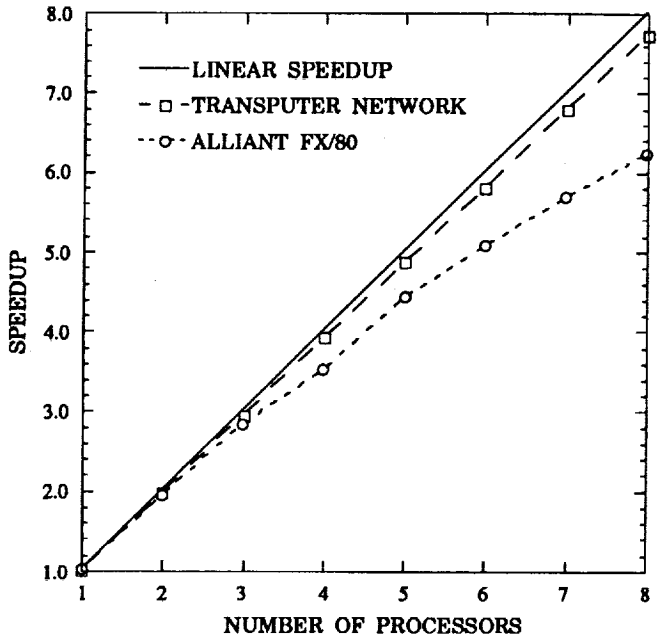

Figure 11. - Speedup comparison between distributed and shared memory architecture computers.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-103671		

4. Title and Subtitle	5. Report Date
Efficient Computation of Aerodynamic Influence Coefficients for Aeroelastic Analysis on a Transputer Network	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
David C. Janetzke and Durbha V. Murthy	E-5883
	10. Work Unit No. 505-63-5B

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington D.C. 20546-0001 and United States Air Force Weapons Laboratory Kirkland AFB Albuquerque, New Mexico 87117-6008	Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

Prepared for the Symposium on Parallel Methods on Large-scale Structural Analysis and Physics Applications, cosponsored by National Aeronautics and Space Administration Langley Research Center and United States Air Force Weapons Laboratory, Hampton, Virginia, February 5-6, 1991.

16. Abstract

Aeroelastic analysis is multi-disciplinary and computationally expensive. Hence, it can greatly benefit from parallel processing. As part of an effort to develop an aeroelastic analysis capability on a distributed memory transputer network, a parallel algorithm for the computation of aerodynamic influence coefficients is implemented on a network of 32 transputers. The aerodynamic influence coefficients are calculated using a 3-dimensional unsteady aerodynamic model and a panel discretization. Efficiencies up to 85-percent were demonstrated using 32 processors. The effects of subtask ordering, problem size and network topology are presented. A comparison to results on a shared memory computer indicates that higher speedup is achieved on the distributed memory system.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parallel processing Unsteady aerodynamics Transputer	Unclassified – Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	30	A03

NASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, Virginia 22161